

Steven M. Hoffberg

From: Steven M. Hoffberg [steve@hoffberg.org]
Sent: Thursday, November 18, 2004 12:17 PM
To: 'Nguyen, Nga'
Subject: 09/599,163 user_db.c

```

/* -----
 * -----
 * registration database access
 * Copyright (c) 1995 Newshare Corporation
 * -----
 */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#ifndef SOLARIS2
#include <strings.h>
#else
#include <string.h>
#define index(a,b) strchr(a,b) /* I cant BELIEVE they dont have this! */
#endif /* SOLARIS2 */
#include <ctype.h>
#include <bstring.h>
#include <sys/types.h>

#include "gdbm.h"
#include "user_db.h"
#include "tvs_log.h"
#include "tvs_config.h"

extern char msgString[];

#ifndef PRIVATE
#define PRIVATE static
#define PUBLIC
#endif /* PRIVATE */

#define DEF_BLOCK 512

typedef datum CONTENTS;          /* "datum" as defined in gdbm.h */
typedef datum KEY;

/* turn on or off syslogd logging */
#ifdef LOGGING
#define LOG_MSG(string) LogMsg(LOG_ERR, string)
#else

```

```

#define LOG_MSG(string) fprintf(stderr, string)
#endif /* LOGGING */

#define DUMP_DELIMIT "____" /* delimited between users on dump */

/* -----
 * handle_fatal - handler for gross gdbm violations
 * -----
 */

PRIVATE void
handle_fatal(char *msg)
{
    extern gdbm_error gdbm_errno;

    sprintf(msgString, "fatal: error %d msg: %s", gdbm_errno, msg);
    LOG_MSG(msgString);

    return;
}

/* -----
 * -----
 * functions for handling the by-name data structure and its internal form
 * -----
 */

/* -----
 * create a reg_profile structure
 * -----
 */

PUBLIC REG_PROFILE
create_reg_profile()
{
    REG_PROFILE reg;

    reg = (REG_PROFILE) malloc(sizeof(struct reg_profile));
    if (!reg) {
        LOG_MSG("create_reg_profile: out of memory");
        return (REG_PROFILE) NULL;
    }

    /* null out the char strings */

    reg->clickshare_username = (char *) NULL;
#ifdef PERSONALIZATION_INFO
    reg->user_first_name = (char *) NULL;
    reg->user_last_name = (char *) NULL;
    reg->user_address_1 = (char *) NULL;

```

```

reg->user_address_2 = (char *) NULL;
reg->user_city = (char *) NULL;
reg->user_state = (char *) NULL;
reg->user_zip = (char *) NULL;
reg->user_country = (char *) NULL;
reg->user_phone_number = (char *) NULL;
reg->user_email_addr = (char *) NULL;
reg->user_access_provider = (char *) NULL;

reg->credit_card_type = (char *) NULL;
reg->credit_card_number = (char *) NULL;
reg->credit_card_expire = (char *) NULL;

reg->demo_release_flag = 0;
reg->demo_age = 0;
reg->demo_gender = (char *) NULL;
reg->demo_years_this_address = 0;
reg->demo_occupation = (char *) NULL;
reg->demo_title = (char *) NULL;
reg->demo_employer = (char *) NULL;
reg->demo_married = 0;
reg->demo_num_children = 0;
reg->demo_num_in_household = 0;
reg->demo_household_income = (char *) NULL;

reg->pref_top_topic = (char *) NULL;
reg->pref_top_city = (char *) NULL;
reg->pref_top_country = (char *) NULL;
reg->pref_int_states = (char *) NULL;
reg->pref_int_region = (char *) NULL;
reg->pref_other_topics = (char *) NULL;
reg->pref_other_global = (char *) NULL;
reg->pref_other_country = (char *) NULL;
#endif /* PERSONALIZATION_INFO */

reg->customer_group = (char *) NULL;
reg->customer_pgn_limit = (char *) NULL;

reg->clickshare_password = (char *) NULL;

reg->pref_parental_discretion = 0;
reg->pref_privacy1 = 0;
reg->pref_premium_charges = 0;
reg->pref_flag4 = 0;
reg->pref_flag5 = 0;
reg->pref_flag6 = 0;
reg->pref_advertising_level = 0;

reg->customer_pgcl_limit = 0;
reg->clickshare_userid = 0;

```

```

    return reg;
}

#ifdef SERIALIZE_COMMA_DELIM
/* -----
 * find out how many bytes in the reg_profile struct (including nulls)
 * -----
 */

PUBLIC int
get_profile_size(REG_PROFILE reg)
{
    int len = 0;

    if (reg == (REG_PROFILE) NULL) return 0;

    /*
     * first the structure itself
     */

    len = sizeof(*reg);

    /*
     * now add space for all the possible strings
     */

#define ADDSTRLEN(len, str) if (str) len += strlen(str) + 1;

    ADDSTRLEN(len, reg->clickshare_username);
#ifdef PERSONALIZATION_INFO
    ADDSTRLEN(len, reg->user_first_name);
    ADDSTRLEN(len, reg->user_last_name);
    ADDSTRLEN(len, reg->user_address_1);
    ADDSTRLEN(len, reg->user_address_2);
    ADDSTRLEN(len, reg->user_city);
    ADDSTRLEN(len, reg->user_state);
    ADDSTRLEN(len, reg->user_zip);
    ADDSTRLEN(len, reg->user_country);
    ADDSTRLEN(len, reg->user_phone_number);
    ADDSTRLEN(len, reg->user_email_addr);
    ADDSTRLEN(len, reg->user_access_provider);

    ADDSTRLEN(len, reg->credit_card_type);
    ADDSTRLEN(len, reg->credit_card_number);
    ADDSTRLEN(len, reg->credit_card_expire);

    ADDSTRLEN(len, reg->demo_gender);
    ADDSTRLEN(len, reg->demo_occupation);
    ADDSTRLEN(len, reg->demo_title);
    ADDSTRLEN(len, reg->demo_employer);
    ADDSTRLEN(len, reg->demo_household_income);

```

```

ADDSTRLEN(len, reg->pref_top_topic);
ADDSTRLEN(len, reg->pref_top_city);
ADDSTRLEN(len, reg->pref_top_country);
ADDSTRLEN(len, reg->pref_int_states);
ADDSTRLEN(len, reg->pref_int_region);
ADDSTRLEN(len, reg->pref_other_topics);
ADDSTRLEN(len, reg->pref_other_global);
ADDSTRLEN(len, reg->pref_other_country);
#endif /* PERSONALIZATION_INFO */
ADDSTRLEN(len, reg->customer_group);
ADDSTRLEN(len, reg->customer_pgn_limit);

ADDSTRLEN(len, reg->clickshare_password);

#undef ADDSTRLEN

return len;
}

/* -----
 * serialize a reg_profile structure
 * Thanks, Michael!
 * -----
 */

PRIVATE char *
serialize_reg_profile(REG_PROFILE reg, int *len)
{
    char *buf, *p;
    REG_PROFILE buf_profile;

    *len = get_profile_size(reg);
    buf = malloc(*len);
    if (!buf) {
        LOG_MSG("out of memory allocating serialized profile buffer");
        return (char *) NULL;
    }

    /* format of buf is:
       copy of the REG_PROFILE reg (buf_profile points here)
       sequence of null-terminated strings
       the (char *) fields in the copy of reg are altered to contain
       offsets from the start of the buffer */

    buf_profile = (REG_PROFILE) buf;
    memcpy (buf_profile, reg, sizeof(*reg));
    p = buf + sizeof(*buf_profile);

    if (!reg->clickshare_username) {
        LOG_MSG("invalid reg_profile received - no username");
    }

```

```

    free(buf);
    return (char *) NULL;
}

/* pack the strings on the end */

#define ADDSTRTOBUF(field) \
if (buf_profile->field) { \
    strcpy(p, buf_profile->field); \
    p += strlen(p) + 1; \
    buf_profile->field -= (int) buf_profile; \
}

    ADDSTRTOBUF(clickshare_username);
#ifdef PERSONALIZATION_INFO
    ADDSTRTOBUF(user_first_name);
    ADDSTRTOBUF(user_last_name);
    ADDSTRTOBUF(user_address_1);
    ADDSTRTOBUF(user_address_2);
    ADDSTRTOBUF(user_city);
    ADDSTRTOBUF(user_state);
    ADDSTRTOBUF(user_zip);
    ADDSTRTOBUF(user_country);
    ADDSTRTOBUF(user_phone_number);
    ADDSTRTOBUF(user_email_addr);
    ADDSTRTOBUF(user_access_provider);

    ADDSTRTOBUF(credit_card_type);
    ADDSTRTOBUF(credit_card_number);
    ADDSTRTOBUF(credit_card_expire);

    ADDSTRTOBUF(demo_gender);
    ADDSTRTOBUF(demo_occupation);
    ADDSTRTOBUF(demo_title);
    ADDSTRTOBUF(demo_employer);
    ADDSTRTOBUF(demo_household_income);

    ADDSTRTOBUF(pref_top_topic);
    ADDSTRTOBUF(pref_top_city);
    ADDSTRTOBUF(pref_top_country);
    ADDSTRTOBUF(pref_int_states);
    ADDSTRTOBUF(pref_int_region);
    ADDSTRTOBUF(pref_other_topics);
    ADDSTRTOBUF(pref_other_global);
    ADDSTRTOBUF(pref_other_country);
#endif /* PERSONALIZATION_INFO */

    ADDSTRTOBUF(customer_group);
    ADDSTRTOBUF(customer_pgn_limit);

    ADDSTRTOBUF(clickshare_password);

```

```

#undef ADDSTRTOBUF

if (p != buf + *len) {
    LOG_MSG("assertion failed in serialize_reg_profile: p != buf + *len");
}

return buf;
}

/* -----
 * un serialize a profile_string into a reg_profile structure
 * -----
 */

PRIVATE REG_PROFILE
unserialize_reg_profile(char *buf, int len)
{
    REG_PROFILE profile;

    profile = (REG_PROFILE) buf;

    /*
     * fix up (char *) entries in struct, which are stored as offsets
     * from start of buffer
     */

#define FIXUPSTR(field) if (profile->field) profile->field += (int) buf;

    FIXUPSTR(clickshare_username);
#ifdef PERSONALIZATION_INFO
    FIXUPSTR(user_first_name);
    FIXUPSTR(user_last_name);
    FIXUPSTR(user_address_1);
    FIXUPSTR(user_address_2);
    FIXUPSTR(user_city);
    FIXUPSTR(user_state);
    FIXUPSTR(user_zip);
    FIXUPSTR(user_country);
    FIXUPSTR(user_phone_number);
    FIXUPSTR(user_email_addr);
    FIXUPSTR(user_access_provider);

    FIXUPSTR(credit_card_type);
    FIXUPSTR(credit_card_number);
    FIXUPSTR(credit_card_expire);

    FIXUPSTR(demo_gender);
    FIXUPSTR(demo_occupation);
    FIXUPSTR(demo_title);
    FIXUPSTR(demo_employer);

```

```
FIXUPSTR(demo_household_income);
```

```
FIXUPSTR(pref_top_topic);
FIXUPSTR(pref_top_city);
FIXUPSTR(pref_top_country);
FIXUPSTR(pref_int_states);
FIXUPSTR(pref_int_region);
FIXUPSTR(pref_other_topics);
FIXUPSTR(pref_other_global);
FIXUPSTR(pref_other_country);
#endif /* PERSONALIZATION_INFO */
```

```
FIXUPSTR(customer_group);
FIXUPSTR(customer_pgn_limit);
```

```
FIXUPSTR(clickshare_password);
```

```
#undef FIXUPSTR
```

```
    return profile;
}
```

```
#else /* SERIALIZE_COMMA_DELIM */
```

```
/* UNFINISHED */
```

```
PRIVATE char *
quote_str(char *p)
{
    char *q, *res;
    int len;

    if (!p)
        p = "";

    for (len = 0, q = p; *q; q++, len++) {
        if ((*q == ',') || (*q == '\\'))
            len++;
    }

    res = q = malloc(len + 1);
    if (!res)
        return NULL;

    while (*p) {
        if ((*p == ',') || (*p == '\\'))
            *q++ = '\\';
        *q++ = *p++;
    }
    *q++ = 0;
```



```

    return res;
}

```

```

PRIVATE char *
unquote_str (char **p)

```

```

{
    char *q, *res;

```

```

    if (!p)
        return NULL;

```

```

    res = q = malloc(strlen(p) + 1);

```

```

    while (*p) {
        if ((*p == '\\') && (*(p+1))) {
            p++;
            *q++ = *p++;
        }
        *q++ = *p++;
    }

```

```

}

```

```

PRIVATE char *

```

```

serialize_reg_profile(REG_PROFILE reg, int *len)

```

```

{
    char *quot_uname, *quot_cust_gp, *quot_pgn_limit, *quot_password;

```

```

    quot_uname = quote_str(reg->clickshare_username);
    quot_cust_gp =

```

```

}

```

```

PRIVATE REG_PROFILE

```

```

unserialize_reg_profile(char *buf, int len)

```

```

{

```

```

}

```

```

#endif /* SERIALIZE_COMMA_DELIM */

```

```

/* -----
 * print the contents of a REG_PROFILE struct to a file
 * -----
 */

```

```

PUBLIC void

```

```

dump_reg_profile(FILE *out , REG_PROFILE reg)

```

```

{
    fprintf(out, "Clickshare-Username: %s\n",
        ((int)reg->clickshare_username ? reg->clickshare_username : "-"));

```

```

#ifdef PERSONALIZATION_INFO

```

```

fprintf(out, "User-First-Name: %s\n",
        ((int)reg->user_first_name ? reg->user_first_name : "-"));

fprintf(out, "User-Last-Name: %s\n",
        ((int)reg->user_last_name ? reg->user_last_name : "-"));

fprintf(out, "User-Address-1: %s\n",
        ((int)reg->user_address_1 ? reg->user_address_1 : "-"));

fprintf(out, "User-Address-2: %s\n",
        ((int)reg->user_address_2 ? reg->user_address_2 : "-"));

fprintf(out, "User-City: %s\n",
        ((int)reg->user_city ? reg->user_city : "-"));

fprintf(out, "User-State: %s\n",
        ((int)reg->user_state ? reg->user_state : "-"));

fprintf(out, "User-Postal-Code: %s\n",
        ((int)reg->user_zip ? reg->user_zip : "-"));

fprintf(out, "User-Country: %s\n",
        ((int)reg->user_country ? reg->user_country : "-"));

fprintf(out, "User-Phone-Number: %s\n",
        ((int)reg->user_phone_number ? reg->user_phone_number : "-"));

fprintf(out, "User-Email-Address: %s\n",
        ((int)reg->user_email_addr ? reg->user_email_addr : "-"));

fprintf(out, "User-Access-Provider: %s\n",
        ((int)reg->user_access_provider ? reg->user_access_provider : "-"));

fprintf(out, "Credit-Card-Type: %s\n",
        ((int)reg->credit_card_type ? reg->credit_card_type : "-"));

fprintf(out, "Credit-Card-Number: %s\n",
        ((int)reg->credit_card_number ? reg->credit_card_number : "-"));

fprintf(out, "Credit-Card-Expire: %s\n",
        ((int)reg->credit_card_expire ? reg->credit_card_expire : "-"));

fprintf(out, "Demo-Release-Flag: %d\n", reg->demo_release_flag);

fprintf(out, "Demo-Age: %d\n", reg->demo_age);

fprintf(out, "Demo-Gender: %s\n",
        ((int)reg->demo_gender ? reg->demo_gender : "-"));

fprintf(out, "Demo-Years-at-Address: %d\n", reg->demo_years_this_address);
fprintf(out, "Demo-Occupation: %s\n",

```

```

        ((int)reg->demo_occupation ? reg->demo_occupation : "-"));
fprintf(out, "Demo-Title: %s\n",
        ((int)reg->demo_title ? reg->demo_title : "-"));
fprintf(out, "Demo-Employer: %s\n",
        ((int)reg->demo_employer ? reg->demo_employer : "-"));
fprintf(out, "Demo-Married: %d\n", reg->demo_married);
fprintf(out, "Demo-Number-Children: %d\n", reg->demo_num_children);
fprintf(out, "Demo-Number-in-Hhld: %d\n", reg->demo_num_in_household);
fprintf(out, "Demo-Household-Income: %s\n",
        ((int)reg->demo_household_income ? reg->demo_household_income : "-"));
#endif /* PERSONALIZATION_INFO */

fprintf(out, "Preference-Parental-Discretion: %d\n",
        reg->pref_parental_discretion);

fprintf(out, "Preference-Privacy1: %d\n", reg->pref_privacy1);
fprintf(out, "Preference-Premium-Charges: %d\n", reg->pref_premium_charges);
fprintf(out, "Preference-Flag4: %d\n", reg->pref_flag4);
fprintf(out, "Preference-Flag5: %d\n", reg->pref_flag5);
fprintf(out, "Preference-Flag6: %d\n", reg->pref_flag6);
fprintf(out, "Preference-Advertising-Level: %d\n",
        reg->pref_advertising_level);

#ifdef PERSONALIZATION_INFO
fprintf(out, "Preference-Top-Topics: %s\n",
        ((int)reg->pref_top_topic ? reg->pref_top_topic : "-"));

fprintf(out, "Preference-Top-City: %s\n",
        ((int)reg->pref_top_city ? reg->pref_top_city : "-"));

fprintf(out, "Preference-Top-Country: %s\n",
        ((int)reg->pref_top_country ? reg->pref_top_country : "-"));

fprintf(out, "Preference-Interest-States: %s\n",
        ((int)reg->pref_int_states ? reg->pref_int_states : "-"));

fprintf(out, "Preference-Interest-Region: %s\n",
        ((int)reg->pref_int_region ? reg->pref_int_region : "-"));

fprintf(out, "Preference-Other-Topics: %s\n",
        ((int)reg->pref_other_topics ? reg->pref_other_topics : "-"));

fprintf(out, "Preference-Other-Global: %s\n",
        ((int)reg->pref_other_global ? reg->pref_other_global : "-"));

fprintf(out, "Preference-Other-Country: %s\n",
        ((int)reg->pref_other_country ? reg->pref_other_country : "-"));
#endif /* PERSONALIZATION_INFO */

fprintf(out, "Customer-Group: %s\n",
        ((int)reg->customer_group ? reg->customer_group : "-"));

```

```

fprintf(out, "Customer-Page-Class-Limit: %d\n", reg->customer_pgcl_limit);

fprintf(out, "Customer-Page-Number-Limit: %s\n",
        ((int)reg->customer_pgn_limit ? reg->customer_pgn_limit : "-"));

fprintf(out, "Clickshare-Password: %s\n",
        ((int)reg->clickshare_password ? reg->clickshare_password : "-"));

fprintf(out, "Clickshare-User-ID: %ld\n", reg->clickshare_userid);
return;
}

```

```

/* -----
 * load a REG_PROFILE struct from a file
 * -----
 */

```

```

PUBLIC REG_PROFILE
load_reg_profile(FILE *in)
{
    char item[1024], *value;
    REG_PROFILE reg;

    reg = create_reg_profile();
    if (!reg) {
        LOG_MSG("load_reg_profile: out of memory!");
        return (REG_PROFILE) NULL;
    }

    while (!feof(in)) {

        /* MIME type colon separation from field and value */

        fgets(item, 1024, in);
        if (item[strlen(item) - 1] == '\n') item[strlen(item)-1] = '\0';

        if (!strcasecmp(DUMP_DELIMIT, item)) break;

        value = index(item, ':');
        if (!value) {
            sprintf(msgString, "load_reg_profile: bad line read {%s}\n", item);
            LOG_MSG(msgString);
            continue;
        }

        *value++ = '\0';          /* terminate item */
        while(isspace(*value)) value++; /* remove blanks */

#define STRMATCH(NAME, PTR) \
        if(!strcasecmp(NAME, item)) {

```

```

    PTR = (char *) malloc(strlen(value) + 1); \
    strcpy(PTR, value); \
}

/* user information */

STRMATCH("Clickshare-Username", reg->clickshare_username);
#ifdef PERSONALIZATION_INFO
    STRMATCH("User-First-Name", reg->user_first_name);
    STRMATCH("User-Last-Name", reg->user_last_name);
    STRMATCH("User-Address-1", reg->user_address_1);
    STRMATCH("User-Address-2", reg->user_address_2);
    STRMATCH("User-City", reg->user_city);
    STRMATCH("User-State", reg->user_state);
    STRMATCH("User-Postal-Code", reg->user_zip);
    STRMATCH("User-Country", reg->user_country);
    STRMATCH("User-Phone-Number", reg->user_phone_number);
    STRMATCH("User-Email-Address", reg->user_email_addr);
    STRMATCH("User-Access-Provider", reg->user_access_provider);

/* credit information */

    STRMATCH("Credit-Card-Type", reg->credit_card_type);
    STRMATCH("Credit-Card-Number", reg->credit_card_number);
    STRMATCH("Credit-Card-Expire", reg->credit_card_expire);

/* demographic information */

    STRMATCH("Demo-Gender", reg->demo_gender);
    STRMATCH("Demo-Occupation", reg->demo_occupation);
    STRMATCH("Demo-Title", reg->demo_title);
    STRMATCH("Demo-Employer", reg->demo_employer);
    STRMATCH("Demo-Household-Income", reg->demo_household_income);
#endif /* PERSONALIZATION_INFO */

    if (strcasecmp("Preference-Parental-Discretion", value))
        reg->pref_parental_discretion = atoi(item);

#ifdef PERSONALIZATION_INFO
    if (strcasecmp("Demo-Release-Flag", value))
        reg->demo_release_flag = atoi(item);
    if (strcasecmp("Demo-Age", value))
        reg->demo_age = atoi(item);
    if (strcasecmp("Demo-Years-at-Address", value))
        reg->demo_years_this_address = atoi(item);
    if (strcasecmp("Demo-Married", value))
        reg->demo_married = atoi(item);
    if (strcasecmp("Demo-Number-Children", value))
        reg->demo_num_children = atoi(item);
    if (strcasecmp("Demo-Number-in-Hhld", value))
        reg->demo_num_in_household = atoi(item);

```

```

/* interest areas */

STRMATCH("Preference-Top-Topics", reg->pref_top_topic);
STRMATCH("Preference-Top-City", reg->pref_top_city);
STRMATCH("Preference-Top-Country", reg->pref_top_country);
STRMATCH("Preference-Interest-States", reg->pref_int_states);
STRMATCH("Preference-Interest-Region", reg->pref_int_region);
STRMATCH("Preference-Other-Topics", reg->pref_other_topics);
STRMATCH("Preference-Other-Global", reg->pref_other_global);
STRMATCH("Preference-Other-Country", reg->pref_other_country);
#endif /* PERSONALIZATION_INFO */

/* service preferences */

if (strcasecmp("Preference-Parental-Discretion", value))
    reg->pref_parental_discretion = atoi(item);
if (strcasecmp("Preference-Privacy1", value))
    reg->pref_privacy1 = atoi(item);
if (strcasecmp("Preference-Premium-Charges", value))
    reg->pref_premium_charges = atoi(item);
if (strcasecmp("Preference-Flag4", value)) reg->pref_flag4 = atoi(item);
if (strcasecmp("Preference-Flag5", value)) reg->pref_flag5 = atoi(item);
if (strcasecmp("Preference-Flag6", value)) reg->pref_flag6 = atoi(item);
if (strcasecmp("Preference-Advertising-Level", value))
    reg->pref_advertising_level = atoi(item);

STRMATCH("Customer-Group", reg->customer_group);
STRMATCH("Customer-Page-Number-Limit", reg->customer_pgn_limit);
STRMATCH("Clickshare-Password", reg->clickshare_password);

if (strcasecmp("Customer-Page-Class-Limit", value))
    reg->customer_pgcl_limit = atoi(item);
if (strcasecmp("Clickshare-User-ID", value))
    reg->clickshare_userid = atoi(item);
}

#undef STRMATCH

return reg;
}

/* -----
 * -----
 * Actions on the by-name database
 * -----
 */

/* -----
 * initialize the databases

```

```

* -----
*/

PUBLIC int
init_databases(int just_do_it)
{
    GDBM_FILE name_dbf;
    GDBM_FILE id_dbf;
    char *name_db, *id_db;

    /* user db */

    name_db = tvs_get_config_param ("RegistrationDB");
    id_db = tvs_get_config_param ("UseridDB");
    if ((!name_db) || (!id_db))
        return 0;

    name_dbf = gdbm_open(name_db, DEF_BLOCK, GDBM_READER,
                        00600, (void (*)())NULL);
    if (name_dbf) {
        if (!just_do_it) {
            sprintf(msgString, "the name database \"%s\" exists already.\n",
                    name_db);
            LogMsg(LOG_NOTICE, msgString);
            return -1;
        }
    }
    if (name_dbf) gdbm_close(name_dbf);
    name_dbf = gdbm_open(name_db, DEF_BLOCK, GDBM_NEWDB,
                        00600, (void (*)())NULL);
    gdbm_close(name_dbf);

    /* group db */

    id_dbf = gdbm_open(id_db, DEF_BLOCK, GDBM_READER,
                        00600, (void (*)())NULL);
    if (id_dbf) {
        if (!just_do_it) {
            sprintf(msgString, "the identifier database \"%s\" exists already.\n",
                    id_db);
            LogMsg(LOG_NOTICE, msgString);
            return -1;
        }
    }
    if (id_dbf) gdbm_close(id_dbf);
    id_dbf = gdbm_open(id_db, DEF_BLOCK, GDBM_NEWDB,
                        00600, (void (*)())NULL);
    gdbm_close(id_dbf);

    return 0;
}

```

```

/* -----
 * open the user-by-name database
 * -----
 */

PUBLIC GDBM_FILE
open_name_db(int rdwr)
{
    GDBM_FILE dbf;
    char *db_name;

    db_name = tvs_get_config_param ("RegistrationDB");
    if (!db_name)
        return 0;

    dbf = gdbm_open(db_name, DEF_BLOCK, rdwr, 00644, handle_fatal);
    return dbf;
}

/* -----
 * add a user to the user-by-name database
 * -----
 */

PUBLIC int
name_add(GDBM_FILE dbf, REG_PROFILE reg)
{
    KEY key;
    CONTENTS cont;
    char *info;
    int err, len;
    extern gdbm_error gdbm_errno;

    if (reg == (REG_PROFILE) NULL) {
        LOG_MSG("NULL registration profile provided");
        return -1;
    }

    if (reg->clickshare_userid == 0) {
        LOG_MSG("bad registration profile provided - no user ID");
        return -1;
    }

    /* create the internal storage format */

    info = serialize_reg_profile(reg, &len);

    /* load up GDBM structs */

    key.dptr = reg->clickshare_username;

```



```

key.dsize = strlen(reg->clickshare_username);

cont.dptr = info;
cont.dsize = len;

err = gdbm_store(dbf, key, cont, GDBM_INSERT);
if (err != 0) {
    sprintf(msgString, "name_add error: %s\n", gdbm_strerror(gdbm_errno));
    LOG_MSG(msgString);
    return -1;
}
return 0;
}

/* -----
 * name_buffer_get
 * -----
 */

PUBLIC char *
name_buffer_get(GDBM_FILE dbf, char *name, int *len)
{
    KEY key;
    CONTENTS cont;
    extern gdbm_error gdbm_errno;
    key.dptr = name;
    key.dsize = strlen(name);

    cont = gdbm_fetch(dbf, key);
    if (cont.dptr == (char *) NULL) {
        sprintf(msgString, "name_get error: %s\n", gdbm_strerror(gdbm_errno));
        LOG_MSG(msgString);
        return (char *) NULL;
    }

    *len = cont.dsize;
    return (char *) cont.dptr;
}

/* -----
 * name_get
 * -----
 */

PUBLIC REG_PROFILE
name_get(GDBM_FILE dbf, char *name)
{
    REG_PROFILE reg;
    char *str;
    int len;

```

```

    str = name_buffer_get(dbf, name, &len);
    if (!str) return (REG_PROFILE) NULL;

    reg = unserialize_reg_profile(str, len);
    return reg;
}

/* -----
 * name_change
 * -----
 */

PUBLIC int
name_change(GDBM_FILE dbf, REG_PROFILE reg)
{
    KEY key;
    CONTENTS cont;
    extern gdbm_error gdbm_errno;
    char *str;
    int err, len = 0;

    if (reg == (REG_PROFILE) NULL) {
        LOG_MSG("NULL registration profile provided");
        return -1;
    }

    if (reg->clickshare_userid == 0) {
        LOG_MSG("bad registration profile provided - no user ID");
        return -1;
    }

    key.dptr = reg->clickshare_username;
    key.dsize = strlen(reg->clickshare_username);

    /* create the internal storage format */

    str = serialize_reg_profile(reg, &len);

    cont.dptr = (char *) str;
    cont.dsize = len;

    err = gdbm_store(dbf, key, cont, GDBM_REPLACE);
    if (err != 0) {
        sprintf(msgString, "name_change error: %s\n", gdbm_strerror(gdbm_errno));
        LOG_MSG(msgString);
        return -1;
    }

    return 0;
}

```

```

/* -----
 * name_exists
 * -----
 */

```

```

PUBLIC int
name_exists(GDBM_FILE dbf, char *name)
{
    KEY key;

    key.dptr = name;
    key.dsize = strlen(name);

    return gdbm_exists(dbf, key);
}

```

```

/* -----
 * name_delete
 * -----
 */

```

```

PUBLIC int
name_delete(GDBM_FILE dbf, char *name)
{
    KEY key;
    int err;
    extern gdbm_error gdbm_errno;

    key.dptr = name;
    key.dsize = strlen(name);

    err = gdbm_delete(dbf, key);
    if (err != 0) {
        sprintf(msgString, "name_delete error: %s\n", gdbm_strerror(gdbm_errno));
        LOG_MSG(msgString);
        return -1;
    }

    return 0;
}

```

```

/* -----
 * name_show
 * -----
 */

```

```

PUBLIC void
name_show(GDBM_FILE dbf, char *name)
{
    KEY key;
    CONTENTS cont;

```

```

REG_PROFILE reg;
extern gdbm_error gdbm_errno;

key.dptr = name;
key.dsize = strlen(name);

cont = gdbm_fetch(dbf, key);
if (cont.dptr == (char *) NULL) {
    sprintf(msgString, "name_show error: %s\n", gdbm_strerror(gdbm_errno));
    LOG_MSG(msgString);
    return;
}

reg = unserialize_reg_profile(cont.dptr, cont.dsize);
if (!reg){
    sprintf(msgString, "name_show: error unserializing database entry");
    LOG_MSG(msgString);
    return;
}

dump_reg_profile((FILE *) stdout, reg);
free(reg);
return;
}

/* -----
 * dump the contents of the user-by-name database to a flat file
 * using a semi-MIME dump format
 * -----
 */

PUBLIC void
name_dump(GDBM_FILE dbf, FILE *out)
{
    KEY key;
    CONTENTS cont;
    REG_PROFILE reg;

    key = gdbm_firstkey(dbf);
    if (!key.dptr) return;

    fprintf(out, "----- User Profiles by Name -----\\n\\n");

    do {
        cont = gdbm_fetch(dbf, key);
        if (cont.dptr) {
            reg = unserialize_reg_profile(cont.dptr, cont.dsize);
            if ((reg) && (reg->clickshare_username[0])) {
                dump_reg_profile(out, reg);
                fprintf(out, "%s\\n", DUMP_DELIMIT);
            }
        }
    }

```

```

        else {
            sprintf(msgString,
                    "name_dump: error getting data from database for %s\n",
                    key.dptr);
            LOG_MSG(msgString);
        }
        free(reg);
    }
    key = gdbm_nextkey(dbf, key);
} while (key.dptr);
}

```

```

/* -----
 * name_close - close the by-name db file
 * -----
 */

```

```

PUBLIC int
close_name_db(GDBM_FILE dbf)
{
    gdbm_sync(dbf);
    gdbm_close(dbf);
    return 1;
}

```

```

/* -----
 * -----
 * actions on the by-id database.
 * -----
 */

```

```

/* -----
 * open the TVS server database
 * -----
 */

```

```

PUBLIC GDBM_FILE
open_id_db(int rdwr)
{
    GDBM_FILE dbf;
    char *db_name;

    db_name = tvs_get_config_param ("UseridDB");
    if (!db_name)
        return 0;

    dbf = gdbm_open(db_name, DEF_BLOCK, rdwr, 00644, handle_fatal);
    return dbf;
}

```

```

/* -----
 * add a TVS server to the TVS name database
 * -----
 */

PUBLIC int
id_add(GDBM_FILE dbf, unsigned long user_id, char *name)
{
    KEY key;
    CONTENTS cont;
    int err;
    extern gdbm_error gdbm_errno;

    key.dptr = (char *) &user_id;
    key.dsize = sizeof(unsigned long);

    cont.dptr = (char *) name;
    cont.dsize = strlen(name);

    err = gdbm_store(dbf, key, cont, GDBM_INSERT);
    if (err != 0) {
        sprintf(msgString, "id_add error: %s\n", gdbm_strerror(gdbm_errno));
        LOG_MSG(msgString);
        return -1;
    }

    return 0;
}

/* -----
 * id_get
 * -----
 */

PUBLIC char *
id_get(GDBM_FILE dbf, unsigned long id)
{
    KEY key;
    CONTENTS cont;

    extern gdbm_error gdbm_errno;
    key.dptr = (char *) &id;
    key.dsize = sizeof(unsigned long);

    cont = gdbm_fetch(dbf, key);
    if (cont.dptr == (char *) NULL) {
        sprintf(msgString, "id_get error: %s\n", gdbm_strerror(gdbm_errno));
        LOG_MSG(msgString);
        return (char *) NULL;
    }
}

```

```

    return (char *) cont.dptr;
}

/* -----
 * get_tvs from TVS database
 * -----
 */

PUBLIC int
id_change(GDBM_FILE dbf, unsigned long user_id, char *name)
{
    KEY key;
    CONTENTS cont;
    extern gdbm_error gdbm_errno;
    int err;

    key.dptr = (char *) &user_id;
    key.dsize = sizeof(unsigned long);

    cont = gdbm_fetch(dbf, key);
    if (cont.dptr == (char *) NULL) {
        sprintf(msgString, "id_change error: %s\n", gdbm_strerror(gdbm_errno));
        LOG_MSG(msgString);
        return -1;
    }

    cont.dptr = (char *) name;
    cont.dsize = strlen(name);

    err = gdbm_store(dbf, key, cont, GDBM_REPLACE);
    if (err != 0) {
        sprintf(msgString, "id_change error: %s\n", gdbm_strerror(gdbm_errno));
        LOG_MSG(msgString);
        return -1;
    }

    return 0;
}

/* -----
 * tvs exists in the TVS database
 * -----
 */

PUBLIC int
id_exists(GDBM_FILE dbf, unsigned long user_id)
{
    KEY key;

    key.dptr = (char *) &user_id;

```

```

    key.dsize = sizeof(unsigned long);

    return gdbm_exists(dbf, key);
}

/* -----
 * id_delete - delete an entry in the by_id database
 * -----
 */

PUBLIC int
id_delete(GDBM_FILE dbf, unsigned long user_id)
{
    KEY key;
    int err;

    key.dptr = (char *) &user_id;
    key.dsize = sizeof(unsigned long);

    err = gdbm_delete(dbf, key);
    if (err != 0) {
        sprintf(msgString, "id_delete error: %s\n", gdbm_strerror(gdbm_erno));
        LOG_MSG(msgString);
        return -1;
    }

    return 0;
}

/* -----
 * id_show - print an entry in the by_id database
 * -----
 */

PUBLIC void
id_show(GDBM_FILE dbf, unsigned long user_id)
{
    KEY key;
    CONTENTS cont;

    key.dptr = (char *) &user_id;
    key.dsize = sizeof(unsigned long);

    cont = gdbm_fetch(dbf, key);
    if (cont.dptr == (char *) NULL) {
        sprintf(msgString, "id_show error: %s\n", gdbm_strerror(gdbm_erno));
        LOG_MSG(msgString);
        return;
    }

    fprintf(stdout, "0x%x %s\n", *key.dptr, cont.dptr);
}

```



```

    free(cont.dptr);
    return;
}

/* -----
 * dump the contents of the by_id database
 * -----
 */

PUBLIC void
id_dump(GDBM_FILE dbf, FILE *out)
{
    KEY key;
    CONTENTS cont;
    int num = 0;

    key = gdbm_firstkey(dbf);
    if (!key.dptr) return;

    fprintf(out, "----- User Names by ID -----\\n\\n");
    do {
        cont = gdbm_fetch(dbf, key);
        if (cont.dptr)
            fprintf(out, "0x%x %s\\n", *key.dptr, cont.dptr);
        else {
            sprintf(msgString, "id_dump: error getting information for ID 0x%x",
                    *key.dptr);
            LOG_MSG(msgString);
        }
        free(cont.dptr);
        key = gdbm_nextkey(dbf, key);
        num++;
    } while (key.dptr);

    fprintf(out, "----- End of User Names by ID (%d users) -----\\n", num);
    return;
}

/* -----
 * close the by-id db file
 * -----
 */

PUBLIC int
close_id_db(GDBM_FILE dbf)
{
    gdbm_sync(dbf);
    gdbm_close(dbf);
    return 1;
}

```

```

/* -----
 * get a full registration, given an ID (two lookups)
 * -----
 */

PUBLIC REG_PROFILE
name_by_id(GDBM_FILE idbf, GDBM_FILE ndbf, unsigned long user_id)
{
    KEY key;
    CONTENTS cont;
    REG_PROFILE reg = (REG_PROFILE) NULL;
    char *nm;

    key.dptr = (char *) &user_id;
    key.dsize = sizeof(unsigned long);

    cont = gdbm_fetch(idbf, key);
    if(cont.dptr) {
        nm = cont.dptr;
        nm[cont.dsize] = '\0';

        reg = name_get(ndbf, (char *) nm);
    }
    return reg;
}

```

Very truly yours,

Steven M. Hoffberg
 Milde & Hoffberg, LLP
 Suite 460
 10 Bank Street
 White Plains, NY 10606
 (914) 949-3100 tel.
 (914) 949-3416 fax
steve@hoffberg.org
www.hoffberg.org

Confidentiality Notice: This message, and any attachments thereto, may contain confidential information which is legally privileged. The information is intended only for the use of the intended recipient, generally the individual or entity named above. If you believe you are not the intended recipient, or in the event that this document is received in error, or misdirected, you are requested to immediately inform the sender by reply e-mail at Steve@Hoffberg.org and destroy all copies of the e-mail file and attachments. You are hereby notified that any disclosure, copying, distribution or use of any information contained in this transmission other than by the intended recipient is strictly prohibited.